Résolution de problèmes de cliques dans les grands graphes

Jocelyn Bernard, Hamida Seba

Résolution de problèmes de cliques dans les grands graphes

Jocelyn BERNARD*, Hamida SEBA*

*Université Lyon 1, LIRIS CNRS 5205, Villeurbanne, France
prenom.nom@univ-lyon1.fr

Résumé. Le problème MCE (Maximal Clique Enumeration) est un des problèmes que l'on rencontre dans l'analyse des graphes de données. C'est un problème NP-difficile pour lequel des solutions adaptées doivent être conçues dans le cas de grands graphes. Nous proposons dans ce papier de répondre à cette problématique en travaillant sur une version compressée du graphe initial. Une telle approche semble intéressante à la fois en terme de temps de calcul et d'espace mémoire. Nous avons donc implémenté notre approche et l'avons comparée à plusieurs solutions de la littérature.

1 Introduction

Un graphe $G = (V, E)$ est un outil de modélisation qui comprend un ensemble de nœuds V ainsi qu'un ensemble d'arêtes E. Une arête est un lien entre deux nœuds. Les graphes ont comme vocation d'être utilisés en tant que structures de données permettant de modéliser des objets ou des problèmes avec de nombreuses applications, notamment dans les domaines des réseaux sociaux, des réseaux informatiques ou encore de la génétique. Parmi ces applications se trouve la recherche de structures, par exemple, dans le cadre de la génétique on peut rechercher des interactions protéines-protéines (Przulj (2003)), on peut également vouloir chercher des communautés dans les graphes issus des réseaux sociaux, etc.

Dans ce travail nous nous intéresserons au problème d'énumération de cliques qui a fait l'objet de plusieurs études : Bron et Kerbosch (1973); Tomita et al. (2006); Prosser (2012). Une clique est un sous-ensemble complet de nœuds du graphe. C'est-à-dire que chacun des nœuds du sous-ensemble est connecté avec l'intégralité des autres.

Le problème d'énumération de cliques maximales ou MCE pour l'anglais Maximal Clique Enumeration est un problème théoriquement NP-Difficile qui se définit de la manière suivante : Une clique K est maximale si elle n'est pas incluse dans une clique de taille supérieure. Le problème d'énumération de cliques maximales consiste à lister l'ensemble des cliques maximales du graphe G.

Dans le cas de grands graphes, ce problème devient difficile à traiter car les données sont nombreuses (le nombre de nœuds et le nombre d'arêtes) et le temps d'exécution est exponentiel avec la taille de ces données. Dans ce cas, plusieurs solutions ont été envisagées : trouver des solutions non exactes mais proches d'un optimal avec des heuristiques Wang et al. (2013), partitionner le graphe Guo et al. (2017), etc. Dans cet article, nous explorons une autre solution qui consiste à travailler sur des données plus petites en compressant le graphe.

La compression de graphes est une opération qui permet la diminution du nombre d'arêtes ou de nœuds du graphe. Cette opération permet par exemple de rendre le graphe plus facile à transmettre ou à lire. Il existe plusieurs méthodes de compression de graphes qui diffèrent selon le type du graphe : simple (Zhou (2015)), orienté (Adler et Mitzenmacher (2001)), étiqueté (Tian et al. (2008)).

Nous proposons de résoudre le problème MCE sur un graphe compressé (sans le décompresser). Nous émettons l'hypothèse que les avantages d'une telle approche sont que :

— le graphe compressé prend moins d'espace mémoire et peut être chargé en mémoire lors de son traitement. Ceci permet de réduire le nombre d'entrées/sorties engendrées si le graphe d'origine ne tient pas en mémoire.
— le graphe compressé étant plus petit, un gain de temps de traitement devrait être réalisé.

Nous avons implémenté cette approche et l'avons comparée à des algorithmes existants. Les résultats sont prometteurs et ouvrent plusieurs perspectives. Le reste de l'article est structuré comme suit : dans la section 2, nous proposons un état de l'art permettant la compréhension de la compression de graphe et des algorithmesexistants pour résoudre le problème MCE. Dans la section 3 nous détaillons notre raisonnement pour la résolution du problème MCE sur le graphe compressé. Dans la section 4, nous proposons une évaluation de notre travail via une expérimentation. Enfin, dans la section 5, nous présentons la conclusion et les perspectives futures qu'offre notre méthode.

2 État de l'art

2.1 La compression de graphes

La décomposition modulaire (Gallai (1967)) va être utilisée pour compresser un graphe. La méthode de la décomposition modulaire revient à réaliser différents agglomérats de nœuds, en fonction de leurs voisinages : McConnell et Spinrad (1999); Habib et Paul (2010). Ces agglomérats sont appelés des modules. Il existe différents types de modules :

— les modules feuilles : chaque module feuille correspondent à un nœud simple dans le graphe original. C'est une feuille de l'arbre de décomposition produit par la compression modulaire.
— les modules séries : un nœud module série correspond à un ensemble de nœuds formant un sous-graphe complet (chacun des nœuds fils du nœud série est relié avec les autres).
— les modules parallèles : un nœud module parallèle correspond à un ensemble de nœuds formant un stable [1]. Il n'existe aucune arête entre les descendants du nœud parallèle. Le complémentaire du sous-graphe le représentant est un graphe complet.
— les nœuds premiers : un module premier correspond à un sous-graphe qui n'est pas complet et dont le graphe complémentaire [2] est également non complet.

La figure 1 illustre la compression d'un graphe en utilisant la décomposition modulaire. L'imbrication des modules, appelée arbre de décomposition modulaire, est présentée dans la figure 2. Les nœuds qui possèdent un voisinage identique sont coloriés de la même manière dans le graphe initial (voir figure 1(a)).

1. Un stable dans un graphe est un ensemble de sommets dont aucun n'est adjacent à un autre
2. Le complémentaire d'un Graphe G est le graphe G' obtenu avec les mêmes nœuds V du graphe G mais dont les arêtes entre deux nœuds sont présentes dans G' uniquement si les deux nœuds ne sont pas adjacents dans G

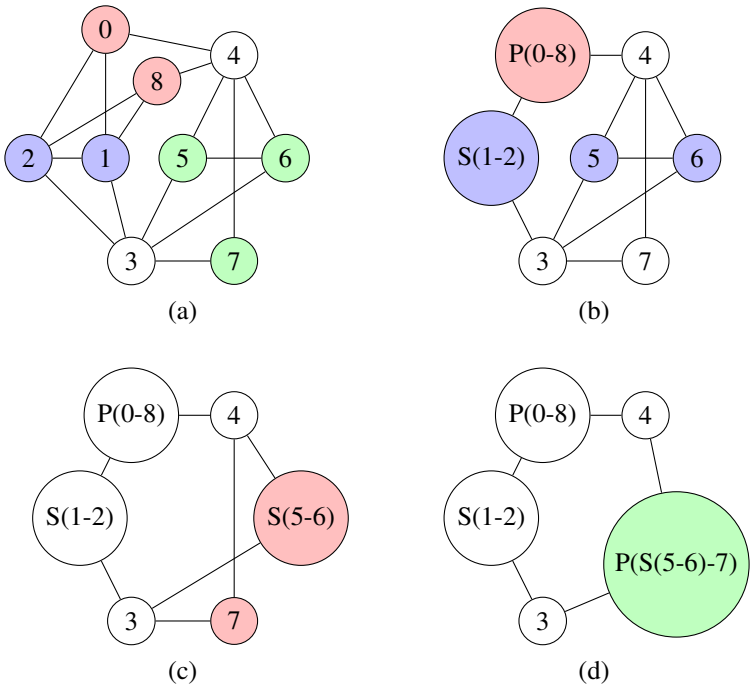

FIG. 1 – *Compression d'un graphe (a) avec deux étapes intermédiaires (b),(c), et le graphe compressé obtenu (d).*

— Les nœuds 1 et 2 ont le même voisinage et sont reliés entre eux. Ils forment un module série $S(1-2)$ (voir figure 1(b)).
— Les nœuds 0 et 8 ont le même voisinage et ne possèdent aucune arête entre eux. Ils forment un module parallèle $P(1-2)$ (voir figure 1(b)).
— Les nœuds 5, 6 et 7 possèdent le même voisinage, cependant ils ne forment ni un stable, ni une clique. Pour compresser cette partie du graphe, il faut d'abord compresser les nœuds 5 et 6 en module série $S(5-6)$, car ils sont reliés et possèdent le même voisinage (voir figure 1(b)/1(c)) puis ensuite de compresser le nœud 7 avec le nœud-module que l'on vient de compresser dans un nœud parallèle $P(S(5-6)-7)$ (voir figure 1(c)/1(d)).

On obtient un alors un graphe partiellement compressé (voir figure 1(d)).

On remarque que le sous-graphe composé des nœuds $P(0-8)$, $S(1-2)$, 3, 4 et $P(S(5-6)-7)$ ne peut plus être compressé par des modules séries ou parallèles. En effet il n'existe pas de nœuds ayant le même voisinage. Nous sommes alors en présence d'un module premier. La décomposition étant finie, on peut construire l'arbre de décomposition (voir figure 2).

2.2 Le problème de cliques MCE

Il existe plusieurs algorithmes et heuristiques pour le problème MCE. Nous décrivons dans cette section une partie des algorithmes proposés dans la littérature.

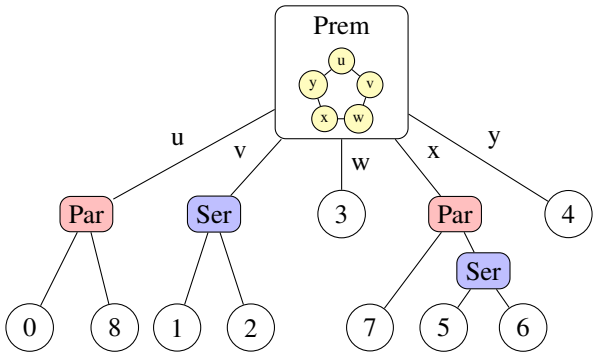

FIG. 2 – *Arbre de décomposition correspondant au graphe de la figure*

2.2.1 MCE

L'algorithme issu de l'article de Bron et Kerbosch (1973) est le premier algorithme créé pour trouver toutes les cliques maximales d'un graphe. Il fonctionne avec 3 ensembles :

— Un ensemble C qui contient la clique partielle construite à un instant t.
— Un ensemble T qui contient les nœuds candidats pour agrandir la clique partielle.
— Un ensemble D qui contient les nœuds ayant déjà été visités pour la construction d'une clique.

L'algorithme initialise donc T avec tous les nœuds V du graphe tandis que C et D sont vides. Ensuite l'algorithme réalise plusieurs itérations. Durant chacune d'elles, il sélectionne un pivot qui l'aide à choisir quels noeuds peuvent être ajoutés à la clique courante. Le choix du pivot permet de trouver toutes les cliques : celles où le pivot en fait partie, car ce ne sont que ses voisins qui sont enlevés de la boucle de sélection et celles où il n'en fait pas partie. La sélection du pivot étant aléatoire, l'expérimentation ne prend pas toujours le même temps et ne retourne pas toujours les cliques dans le même ordre.

2.2.2 Tomita

L'algorithme de Tomita et al. (2006) est aujourd'hui l'algorithme de référence. Sa complexité est de l'ordre de O($3^{(n/3)}$). Il diffère de la recherche de Bron et Kerbosh par le choix du pivot qui permet un élagage généralement plus efficace. Il utilise lui aussi 3 ensembles :

— Un ensemble Q qui contient la clique partielle qui est construite à un instant t.
— Un ensemble $SUBG$ et un ensemble $CAND$ qui permettent de sélectionner le meilleur pivot possible, puis le meilleur candidat, dans le but d'agrandir la clique.

L'algorithme initialise l'ensemble Q à vide tandis que $SUBG$ et $CAND$ sont initialisés avec l'ensemble des nœuds V. L'élagage se fait par le choix du pivot. Ce choix permet une recherche plus rapide car il maximise la taille de l'ensemble $CAND \cap \Gamma(pivot)$, où $\Gamma(pivot)$ correspond au voisinage du pivot. De plus si l'ensemble $CAND$ est vide mais pas $SUBG$ on retourne en arrière car la clique courante C générée peut être un sous ensemble d'une clique maximale. La procédure est présenté dans Algorithme 1.

Algorithm 1 Procedure-Tomita(Q,$SUBG$,$CAND$

if $SUBG = \emptyset$ **then**
  Retourner Q comme clique maximale
end if
choisir pivot $u \in SUBG$ qui maximise $|CAND \cap \Gamma(u)|$
while $CAND - \Gamma(u) \neq \emptyset$ **do**
  choisir $q \in CAND - \Gamma(u)$
  $Q \leftarrow Q + q$
  Procedure-Tomita($Q, SUBG \cap \Gamma(u), CAND \cap \Gamma(u)$)
  $Q \leftarrow Q - q$
  $CAND \leftarrow CAND - q$
end while

2.2.3 Eppstein

Eppstein et Strash (2011) ont proposé une amélioration de l'algorithme de Tomita. Avant d'exécuter Tomita, ils proposent d'utiliser la dégénérescence de graphe [3]. Pour chaque graphe G, on peut calculer son ordre de dégénérescence, qui est un ordre linéaire des sommets tels que chaque sommet a au plus d voisins plus tard dans l'ordre. La dégénérescence d'un graphe et son ordre peuvent être calculés en temps linéaire O(m) Batagelj et Zaversnik (2003). L'algorithme utilise les mêmes ensembles que Tomita, mais ils sont initialisés dans l'ordre de dégénérescence.

2.2.4 RMC

Wang et al. (2013) proposent quant à eux de lister seulement une partie des cliques maximales d'un graphe en évitant de retourner des cliques qui se chevauchent, c'est-à-dire dont les identifiants des nœuds qui les composent sont quasiment les mêmes. Pour cela, ils proposent la notion de τ-visibilité. La τ-visibilité est une variable qui permet d'écarter des cliques qui sont proches de la clique trouvée précédemment selon le seuil défini par l'utilisateur.

Les auteurs déclinent également l'algorithme en deux versions : une qui applique strictement le taux τ et une version aléatoire qui l'applique à quelques nuances près. L'algorithme ressemble à celui de Tomita et d'Eppstein mais sort plus tôt de la fonction si la clique qui est en train d'être construite se trouve être trop similaire à la précédente.

3 Énumération de cliques maximales sur un graphe compressé

L'algorithme que nous proposons est un algorithme récursif qui part de la racine de l'arbre de décomposition modulaire et qui l'explore en profondeur. Les cliques sont relevées de manières différentes en fonction du type du nœud parcouru.

3. La *dégénérescence* d'un graphe G est le plus petit nombre k de telle sorte que chaque sous-graphe $S \in G$ contient un sommet de degré au plus k

— Si le nœud est une feuille on retourne uniquement l'identifiant correspondant au nœud dans le graphe initial.
— Si le nœud est de type série : étant donné que tous ses fils sont reliés entre eux (et forment donc un graphe complet), on retourne un ensemble de listes d'identifiants qui correspondent aux nœuds pouvant être compris dans la clique trouvée. La figure 3 illustre un exemple.
— Si le nœud est de type parallèle : étant donné qu'aucun de ses fils n'est relié avec un autre (le complémentaire est premier), on retourne un ensemble de listes d'identifiants où chaque liste est issue du retour des nœuds fils du module parallèle. La figure 4 donne un exemple.
— Si le nœud est de type premier, il est nécessaire de lancer une recherche de clique. On commence alors par construire le graphe compressé correspondant $Gc = (V_c, E_c)$ à l'aide de ses nœuds fils V_c et de leurs arêtes E_c issues des listes de voisinages créés lors de la compression du graphe initial. On exécute ensuite un algorithme de recherche de cliques maximales sur le graphe compressé qui est basé sur Tomita. Ensuite, on parcourt les cliques trouvées de la manière suivante : on explore chacun des nœuds-module de la clique et on remplace leur identifiant de module par la liste de nœuds qu'ils retournent lors de leur appel de la fonction de recherche (qui dépend du type du module). La figure 5 montre un exemple pour ce cas.

3.1 Optimisation

Afin d'améliorer la vitesse des algorithmes, nous avons ajouté une optimisation. Cette optimisation permet à un nœud-module dont le type est différent du module feuille et dont le père est de type premier de sauvegarder la valeur qu'il a renvoyé au premier appel. Cela permet de ne pas avoir à parcourir à nouveau tout son sous-arbre s'il fait à nouveau appel à ce nœud. En effet si le père est de type premier, il est possible que ce nœud se retrouve dans plusieurs cliques et qu'il soit donc appelé plusieurs fois. Par exemple si un nœud-module est lui-même premier et qu'il se trouve dans deux cliques différentes lorsque l'on évalue les cliques dans le graphe compressé de son père, on va appeler deux fois sa fonction de retour. Or, lors du second appel on va devoir à nouveau reconstruire un graphe, lancer la recherche de clique et reconstruire la liste de retour, ce qui peut être coûteux en temps. Nous choisissons donc d'enregistrer la liste de retour et de retourner cette valeur en cas de second appel.

4 Évaluation

Dans cette section nous allons décrire les expérimentations réalisées afin d'évaluer notre approche. Pour cela nous avons implémenté notre algorithme ainsi que les algorithmes que nous avons présentés dans la partie état de l'art. Nous avons implémenté tous les algorithmes en C++ à l'aide la librairie SNAP[4]. Ceci afin d'utiliser les mêmes structures pour l'ensemble des algorithmes. Nous avons travaillé sur une machine 64 bits avec un système d'exploitation Ubuntu v14.04 LTS, un processeur i5-4690 de fréquence 3.5GHz et une mémoire vive de 8G.

Les algorithmes ont été compilés avec la version 4.8.2 du compilateur g++ et les options -std=c++11 et l'optimisation -03.

4. https ://snap.stanford.edu/

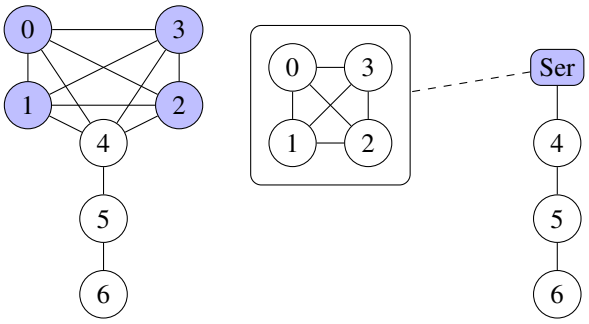

FIG. 3 – *Exemple du problème MCE avec un nœud série : Le nœud de type série, lorsqu'il est interrogé dans la fonction de recherche de cliques retourne la liste de ses nœuds fils (0,1,2 et 3). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {5,6}, {4,5} et {0,1,2,3,4}.*

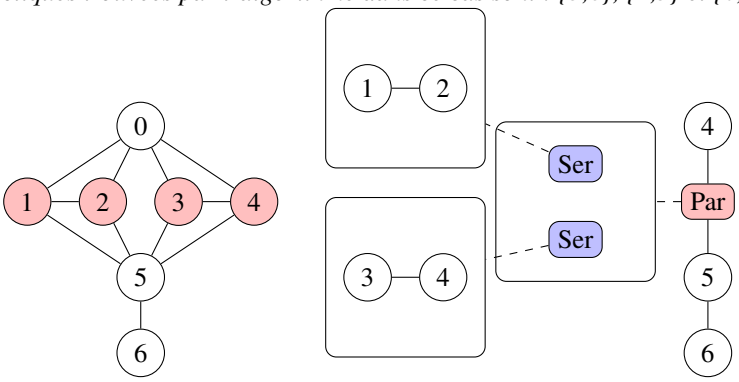

FIG. 4 – *Exemple du problème MCE avec un nœud parallèle : Le nœud de type parallèle, lorsqu'il est interrogé dans la fonction de recherche de cliques retourne un ensemble de listes issues des listes de retour de ses nœuds fils, dans ce cas il retourne ({1,2},{3,4}). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {5,6}, {0,1,2},{0,3,4},{5,1,2} et {5,3,4}.*

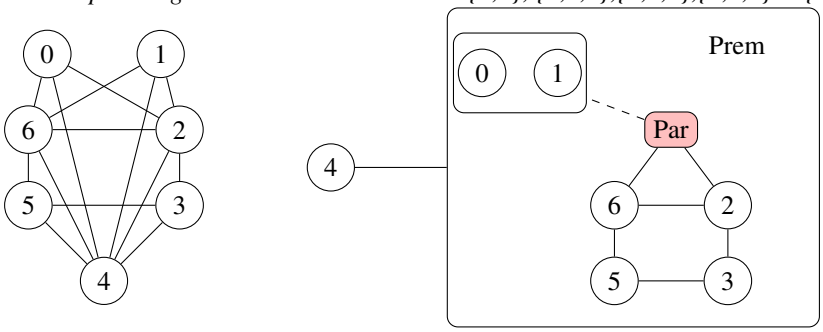

FIG. 5 – *Exemple du problème MCE avec un nœud premier : La racine de l'arbre est un nœud série qui relie le nœud 4 à un nœud premier dont on peut observer le graphe compressé correspondant à droite. Notre algorithme va donc lancer une recherche de clique maximale dans le graphe compressé et retourner une liste de cliques issue de sa recherche, à savoir ({0,2,6},{1,2,6},{2,3},{3,5},{5,6}). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {0,2,4,6},{1,2,4,6},{2,3,4},{3,4,5} et {4,5,6}.*

Notre algorithme de compression est basé sur l'algorithme de décomposition modulaire de Montgolfier (2003) qui a une complexité en O(m $\log$ n). L'algorithme de décomposition est lui-même issu de la concaténation de deux algorithmes distincts :

1. Le premier réalise une permutation factorisante des sommets Capelle et al. (2002) du graphe Capelle (1997) grâce à une technique d'affinage des partitions Habib et al. (1999).
2. La seconde construit l'arbre de décomposition modulaire à l'aide la permutation issue du premier algorithme Bergeron et al. (2008).

De plus, afin de permettre aux algorithmes de recherche de cliques de pouvoir fonctionner, nous avons ajouté un troisième algorithme qui ajoute aux nœuds fils d'un graphe premier une liste de voisinage. Concrètement, cela signifie que si un nœud est voisin avec un autre nœud dans le graphe initial, les modules les représentants dans un graphe premier auront également une arête de voisinage. L'algorithme est un algorithme récursif qui se contente de parcourir l'arbre en ajoutant une arête lorsqu'elle était existante dans le graphe précédent. Il teste chacun des descendants d'un nœud premier pour savoir si il existe une arête entre les nœuds qu'ils représentent dans le graphe original et ajoute une arête entre les descendants le cas échéant. L'algorithme 2 présente le détail de cette opération.

Algorithm 2 creeVoisins(Module m)

```
1:  if m est de type premier then
2:    for i allant de 0 à |m− > descendant| do
3:      for j allant de i à |m− > descendant| do
4:        if i est voisin de j dans le graphe then
5:          Ajouter i à la liste de voisinage de j dans le graphe compressé
6:          Ajouter j à la liste de voisinage de i dans le graphe compressé
7:        end if
8:      end for
9:    end for
10: end if
11: if m a des descendants premiers then
12:   for s ∈ m− > descendant do
13:     creeVoisins(s)
14:   end for
15: end if
```

4.1 Graphes de test

Le tableau 1 présente les graphes sur lesquels nous avons testé les algorithmes. Pour chaque graphe nous donnons dans le tableau 2 le temps nécessaire pour le compresser ainsi que la taille du graphe compressé (le nombre de modules), le nombre total d'arêtes dans l'ensemble des nœuds premiers de l'arbre et enfin ce que nous avons appelé le taux de compression qui correspond au rapport nombre d'arêtes dans le graphe compressé sur le nombre d'arêtes dans le graphe initial. Ces graphes sont tirés des bases de données des projets Dimacs et String (*http ://dimacs.rutgers.edu/Challenges/* & *https ://string-db.org/*).

ID	**Nom**	|V|	|E|	densité	d_{Min}	d_{Max}	d_{Moy}	Taille
(1)	16pk	4919	4972	0.000411	1	4	2.021	51.2Ko
(2)	3djd	11862	12010	0.000171	1	4	2.025	131Ko
(3)	c-fat500-5	500	23191	0.185900	92	95	92.764	194Ko
(4)	6pfk	17123	17263	0.000118	1	4	2.016	198Ko
(5)	Caenorhabditis_elegans	6173	26184	0.001374	1	418	8.483	273Ko
(6)	Takifugu_rubipres	5872	27077	0.001571	1	162	9.222	281Ko
(7)	3dmk	30386	30773	0.000067	1	4	2.025	369Ko
(8)	c-fat500-10	500	46627	0.373764	185	188	186.508	390Ko
(9)	Drosophila_melanogaster	8624	39466	0.001061	1	311	9.153	413Ko
(10)	Rattus_norvegicus	8763	39932	0.001040	1	334	9.114	419Ko

TAB. 1 – *Présentation des graphes de tests*

Nom	nombre modules	nombre d'arêtes	temps	taux
16pk	5532	4199	0.112408	15.547064
3djd	13488	9976	0.118331	16.935887
c-fat500-5	517	16	0.012088	99.931008
6pfk	19467	14226	0.412102	17.592539
Caenorhabditis_elegans	6656	24570	0.252191	6.164070
Takifugu_rubipres	6651	23530	0.248460	13.099679
3dmk	34537	25468	1.621132	17.239138
c-fat500-10	509	8	0.017551	99.982843
Drosophila_melanogaster	9137	38297	0.656012	2.962043
Rattus_norvegicus	9517	37279	0.611329	6.643794

TAB. 2 – *Présentation de la compression des graphes*

4.2 Résultats et discussions

Nous avons lancé des algorithmes sur l'ensemble des graphes. Le tableau 3 présente les résultats. La figure 4.2 présente les différents temps de calcul. Le premier graphique représente les résultats des algorithmes présentés dans la section état de l'art. Pour RMC, les résultats présentés sont ceux où l'algorithme a réalisé le plus petit temps d'exécution. Il convient également de noté que la courbe des 2 Eppstein étant très proche on ne voit que Eppstein-Random. Les deux autres graphiques représentent les différents temps de calcul entre Tomita (le meilleur algorithme exact pour le problème MCE) et notre algorithme sur le graphe compressé. Les graphiques présentés sont ceux dont les graphes ont un taux de compression de l'ordre de 0 à 20% pour le graphe de gauche et de l'ordre de 99% pour le graphe de droite. Nous avons également rajouté une courbe représentant le temps de notre algorithme plus le temps de compression du graphe.

Pour le problème d'énumération de cliques, même en ajoutant le temps de compression de graphe, on remarque que notre algorithme fait mieux que Tomita, notre algorithme de référence, dans les cas où le graphe a un taux de compression supérieur à 15%. Les autres algorithmes tels que MCE et RMC donnent certaines fois de meilleurs résultats en terme de temps, c'est pourquoi il serait intéressant de les adapter sur le graphe compressé.

4.3 Étude de la complexité

Nous conjecturons que la complexité de l'algorithme d'énumération de cliques est de l'ordre de $\sum_{i=1}^{ms} n_i + \sum_{j=1}^{mp} n_j + \sum_{k=1}^{mr} 3^{n_k/3} + n$ où *ms*, *mp* et *mr* représentent respectivement le nombre de modules séries, parallèles et premiers et n_i,n_j et n_k représentent respectivement le nombre de nœuds fils de chacun d'entre eux. n est le nombre de nœuds dans le graphe initial.

Algorithme/Graphe	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
MCE	4972 0.1913	12010 1.3329	16 0.1874	17263 3.2995	19386 2.3920	18877 2.4517	30773 7.331336	8 0.5963	30613 3.782695	41118 4.7760
Tomita	4972 0.5277	12010 3.0566	16 0.0675	17263 6.4554	19386 1.2426	18877 1.3051	30773 19.9444	8 0.2033	30613 2.5967	41118 2.7588
Eppstein	4972 0.8198	12010 5.0202	16 0.0612	17263 11.0183	19386 1.7087	18877 1.5647	30773 37.3132	8 0.1663	30613 3.0690	41118 3.6712
Eppstein-hybrid	4972 0.8198	12010 5.0214	16 0.0613	17263 11.0178	19386 1.7068	18877 1.5779	30773 37.3035	8 0.1654	30613 3.0764	41118 3.6775
RMC(0.5)	4968 0.0114	12006 0.0297	13 0.2491	17259 0.0428	13432 0.3209	13701 0.3013	30769 0.0777	5 0.5832	23700 0.4547	22536 0.6302
RMC-random(0.5)	2028 0.0078	4447 0.0200	5 0.2250	6754 0.0294	8620 0.2571	8818 0.2314	12254 0.0530	2 0.6158	16476 0.3733	13868 0.4763
RMC(0.7)	4968 0.0115	12006 0.0303	14 0.2520	17259 0.0425	14102 0.3447	14449 0.3242	30769 0.0762	6 0.5233	24663 0.4869	28098 0.7853
RMC-random(0.7)	3209 0.0093	7435 0.0249	7 0.2468	10778 0.0352	10789 0.2867	11020 0.2780	19422 0.0631	4 0.5866	19755 0.4206	19912 0.6239
RMC(0.9)	4968 0.0110	12006 0.0300	14 0.2599	17259 0.0432	15213 0.3701	15620 0.3514	30769 0.0765	6 0.5271	26588 0.5358	38843 0.9258
RMC-random(0.9)	4351 0.0107	10459 0.0290	11 0.2412	15108 0.0417	13579 0.3484	13977 0.3318	26945 0.0732	6 0.5261	24106 0.5087	32121 0.8280
RMC(1.0)	4968 0.0115	12006 0.0312	14 0.2929	17259 0.0444	15370 0.3732	15836 0.3576	30769 0.0803	6 0.7644	27120 0.5467	39223 0.9402
RMC-random(1.0)	4968 0.0111	12006 0.0306	14 0.2591	17259 0.0437	15370 0.3744	15836 0.3559	30769 0.0793	6 0.5277	27120 0.5464	39223 0.9305
AlgorithmeCompresse	4972 0.4198	12010 0.2600	16 0.0002	17263 1.2894	19386 1.4125	18877 1.2926	30773 5.6461	8 0.0003	30613 3.4481	41118 4.4114

TAB. 3 – *Résultats pour le problème MCE. La première ligne donne le nombre de cliques trouvé par l'algorithme et la seconde son temps (en secondes)*

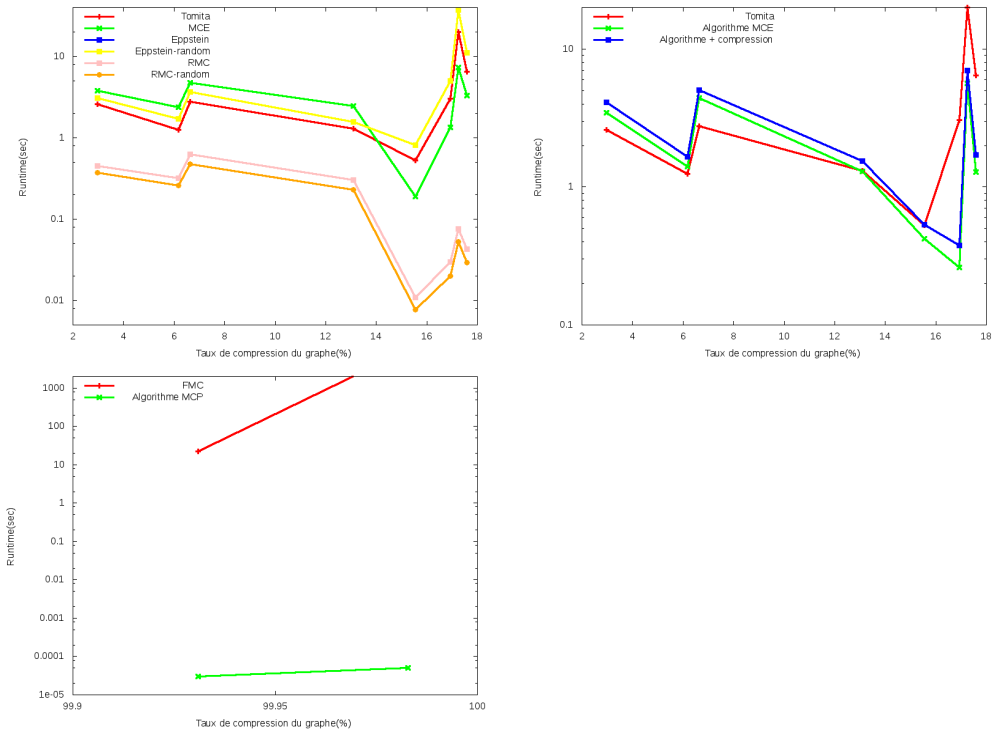

FIG. 6 – *Graphiques montrant les temps de calculs des algorithmes en fonction du taux de compression des graphes.*

5 Conclusion et ouvertures

Le problème d'énumération de cliques maximales dans les graphes est un problème complexe et difficile. Ce problème devient d'autant plus difficile lorsque les graphes sur lesquels nous travaillons sont grands. Pour résoudre ces problèmes, il existe de nombreux algorithmes et heuristiques. Dans ce papier, nous avons proposé et évalué une nouvelle méthode pour résoudre ce problème.

Notre algorithme se déroule en deux phases. La première consiste à compresser le graphe, via la décomposition modulaire pour diminuer la taille de celui-ci. La décomposition modulaire est une méthode de compression qui nous permet de garder les informations nécessaires à la recherche de cliques. La deuxième phase consiste à chercher les cliques sur le graphe compressé.

L'expérimentation réalisée démontre que notre algorithme d'énumération de cliques montre de meilleures performances dès que la compression du graphe est supérieure à 15%. Il nous semble intéressant de continuer le travail commencé en poursuivant plusieurs options :

— Augmenter l'efficacité des algorithmes en ajoutant des informations lors de la compression de graphes, notamment sur la taille des cliques dans les modules séries et parallèles.
— Développer de nouveaux algorithmes sur les graphes compressés.
— Regarder la consommation de mémoire vive de l'algorithme.
— Adapter la méthode aux autres types de graphes tels que les graphes orientés.
— Proposer une méthode de compression et de recherche de cliques en parallèle, l'algorithme de recherche de cliques démarrant dès qu'une compression du graphe est suffisante.

Références

Adler, M. et M. Mitzenmacher (2001). Towards compressing web graphs. In *Data Compression Conference, 2001. Proceedings. DCC 2001.*, pp. 203–212. IEEE.

Batagelj, V. et M. Zaversnik (2003). An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*.

Bergeron, A., C. Chauve, F. De Montgolfier, et M. Raffinot (2008). Computing common intervals of k permutations, with applications to modular decomposition of graphs. *SIAM Journal on Discrete Mathematics 22*(3), 1022–1039.

Bron, C. et J. Kerbosch (1973). Algorithm 457 : finding all cliques of an undirected graph. *Communications of the ACM 16*(9), 575–577.

Capelle, C. (1997). *Decompositions de graphes et permutations factorisantes*. Ph. D. thesis, Montpellier 2.

Capelle, C., M. Habib, et F. De Montgolfier (2002). Graph decompositions and factorizing permutations. *Discrete Mathematics and Theoretical Computer Science 5*(1), 55–70.

Eppstein, D. et D. Strash (2011). Listing all maximal cliques in large sparse real-world graphs. In *Experimental Algorithms*, pp. 364–375. Springer.

Gallai, T. (1967). Transitiv orientierbare graphen. *Acta Mathematica Hungarica 18*(1), 25–66.

Guo, J., S. Zhang, X. Gao, et X. Liu (2017). Parallel graph partitioning framework for solving the maximum clique problem using hadoop. In *Big Data Analysis (ICBDA), 2017 IEEE 2nd International Conference on*, pp. 186–192. IEEE.

Habib, M. et C. Paul (2010). A survey of the algorithmic aspects of modular decomposition. *Computer Science Review 4*(1), 41–59.

Habib, M., C. Paul, et L. Viennot (1999). Partition refinement techniques : An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science 10*(02), 147–170.

McConnell, R. M. et J. P. Spinrad (1999). Modular decomposition and transitive orientation. *Discrete Mathematics 201*(1), 189–241.

Montgolfier, F. d. (2003). *Décomposition modulaire des graphes : théorie, extensions et algorithmes*. Ph. D. thesis, Montpellier 2, Université des Sciences et Techniques du Languedoc.

Prosser, P. (2012). Exact algorithms for maximum clique : A computational study. *Algorithms 5*(4), 545–587.

Przulj, N. (2003). Graph theory approaches to protein interaction data analysis. *proteins 120*, 000.

Tian, Y., R. A. Hankins, et J. M. Patel (2008). Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 567–580. ACM.

Tomita, E., A. Tanaka, et H. Takahashi (2006). The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science 363*(1), 28–42.

Wang, J., J. Cheng, et A. W.-C. Fu (2013). Redundancy-aware maximal cliques. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 122–130. ACM.

Zhou, F. (2015). Graph compression. *Department of Computer Science and Helsinki Institute for Information Technology HIIT*, 1–12.

Summary

The MCE (Maximal Clique Enumeration) is a problem encontred in data graph analyses and mining. However, this problem is NP-hard. Consequently, appropriate solutions must be proposed in the case of large graphs. We propose in this work to answer this problem by working on a compressed version of the original graph. This approach seems interesting both in terms of computation time and memory space. So we implemented our approach and we compared it with several solutions from the literature.